A report that considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics

Niamh Belton

MEASURING THE software engineering process

**Section 1:**

**1.1 Introduction:**

**What does measuring and assessing the Software Engineering process mean?**

The idea of measuring the software engineering process is as old as software engineering itself. The software engineering process can be measured using ‘Software Metrics’. Metrics refer to the measurements/data that are taken throughout the software engineering process.

There are 2 types of Software metrics;

1. Product metrics – metrics that measure the cost and quality of the product and includes factors such as code length, complexity, reusability and maintainability
2. Process metrics – metrics that measure the efficiency of the processes involved in the development of the software such as editing time.

(Sillitti, Janes, Succi and Vernazza, n.d.)

The concept of software Metrics can then be broken down into 3 areas;

1. It refers to the attempt to quantify the quality of the software in measureable data that can then be used to assess the software and the progress in developing the software.
2. It also refers to models that are able to predict the quality and performance of the software.
3. The subject of testing and assessing the quality of software by recording inefficiencies or bugs during development.

(Fenton and Neil, 1999).

**1.2 The need to measure the software engineering process:**

**Costs** - Measuring the software engineering process is important from the aspects of costs and time. Software engineering depends hugely on human effort. Therefore, the main cost associated with the development of software is the time spent. Measuring the software engineering process and predicting possible failures allows the engineer to iteratively change and improve his work during the process rather than finishing his software, realizing his mistakes or inefficiencies and starting over. This would would incur huge costs and effort.

**Quality -**. Software metrics allow software practitioners to predict failures, bugs and overall quality of their software. With this knowledge they can ensure a better quality final software product. Quality of software is important when developing software for medical uses or security systems.

By measuring the processes of software engineering of the top software engineers, this will allow other software engineers to adopt the best process of developing the software. Hence, ensuring reduced time costs and a higher quality software (Sillitti, Janes, Succi and Vernazza, n.d.).

**1.3 What Software Engineering data is measurable and why is this data useful?**

**Lines of Code measure:**

The first practice of measuring the software engineering process was the Lines of Code Measure (LOC). The method simply involved recording the number of lines of code a programmer would write per month. The KLOC was a measure of thousands of lines of code. This method was thought of as a way of measuring the productivity of the programmer. By using LOC to measure the productivity of the programmer, costs such as time and effort of the software development can be calculated. Hence, software engineers and researchers began using LOC for resource predication models, predicting the effort and cost of developing the software. They built these models on the idea that the effort was a function of the LOC (Fenton and Neil, 1999).

**Limitations to LOC:**

While the LOC remains a key aspect of measuring the software engineering process, there are disadvantages to using it to measure the quality of developing software;

1. LOC cannot not assess the development of software across different programming languages. The LOC in a basic language can not compare to the functionality, quality, effort/cost and other notions of product to a LOC of a high-level language (Fenton and Neil, 1999).
2. By using LOC to measure programmer productivity, it could cause the programmer to sacrifice quality in order to have an increased lines of code to appear more productive (Sommerville, 2008).

**Bugs per line of code:**

The LOC and KLOC then began to be used to measure the number of defects per thousand lines of codes. This was thought of as a method for recording the quality of the program (Fenton and Neil, 1999).

**Note:** In 1971, Akiyama was the first to attempt to use Software Metrics such as LOC and KLOC to **predict** the quality of the software (Fenton and Neil, 1999). Previous to this, software metrics were used simply to assess the software development progress. This was the beginning of using prediction models to predict the quality of code. His model was a regression based model that measured ‘defect density’, number of defects per KLOC (Stepford, 2017).

**Commits/Check-ins:** The number of commits a developer makes can be used to measure his productivity.

**Time Tracking:** Lead-time – the length of time it takes to develop the software. Cycle time – the length of time it takes to make changes to the software.

**Pull requests:** The number of pull requests a developer makes can be used to measure his productivity.

**Quantifying Happiness:**

Quantifying the happiness of the programmer can be considered a way of measuring the software engineering process. Studies have shown there is a direct correlation between the happiness of a worker and their productivity. Happier people perform significantly better than unhappy people. The question is how does one quantify happiness. The company Hitachi developed a way for quantifying happiness by studying the relationship between physical activity and happiness by using wearable technology. They have recorded over a million days of data on people’s physical activities. They look for correlations between happiness and patterns in physical activity. From this they created the “1/T rule” which can be used to quantify happiness (Yano et al., 2015). By quantifying a developer’s happiness, you can measure his productivity which will in turn establish the costs and time associated with the development process.

**The Personal Software Process (PSP)**

The personal Software Process is a method of tracking the software’s development as the engineer works on it. It allows the engineer to monitor their performance, keep the development process on schedule and analyze the software at any point during the development (Sillitti, Janes, Succi and Vernazza, n.d.). The PSP method is of a manual nature and easy to use. There are 3 phases associated with the PSP, the developer entering their data, data collection and data analysis.

**1) Entering the Data**

The PSP requires that the engineer enters a collection of data using PSP methodology. Forms are an easy and convenient way for software practitioners to record information. Forms specify what information they need and where exactly to record it (Resources.sei.cmu.edu, n.d.). Example of forms are listed below;

* a project plan summary
* a time-recording log
* a defect-recording log
* a process improvement proposal
* a size estimation template
* a time estimation template
* a design checklist
* a code checklist.

(Johnson, 2013)

**2) Data Collection**

PSP tracks and monitors the process acquiring data such as time, size and defects information.

**3) Data Analysis**

* At any point during the software process, the PSP can provide the data and make suggestions on how to improve the process. It also reveals problems with the software and makes estimates of timing and quality of the software.
* The users of the PSP are required to analyze the process data themselves. The user can do a ‘postmortem’ of the PSP data.
* The data can be used for aiding future projects
* Users can analyze their data from historical data
* They can also analyze themselves on a personal level by reviewing their ability to meet time targets and where they expected their software to be at a stage in time versus where it actually is.

(Resources.sei.cmu.edu, n.d.).

**Note:**

Watson Humphrey created the PSP and intended for the users of the PSP to analyze their own data. As the data involved judgement and was specific to each software engineer, he thought it was unlikely that any toolkit would exist that would gather and analyze the information for the software engineer (Johnson, 2013).

**Section 2:**

**Computational Platforms**

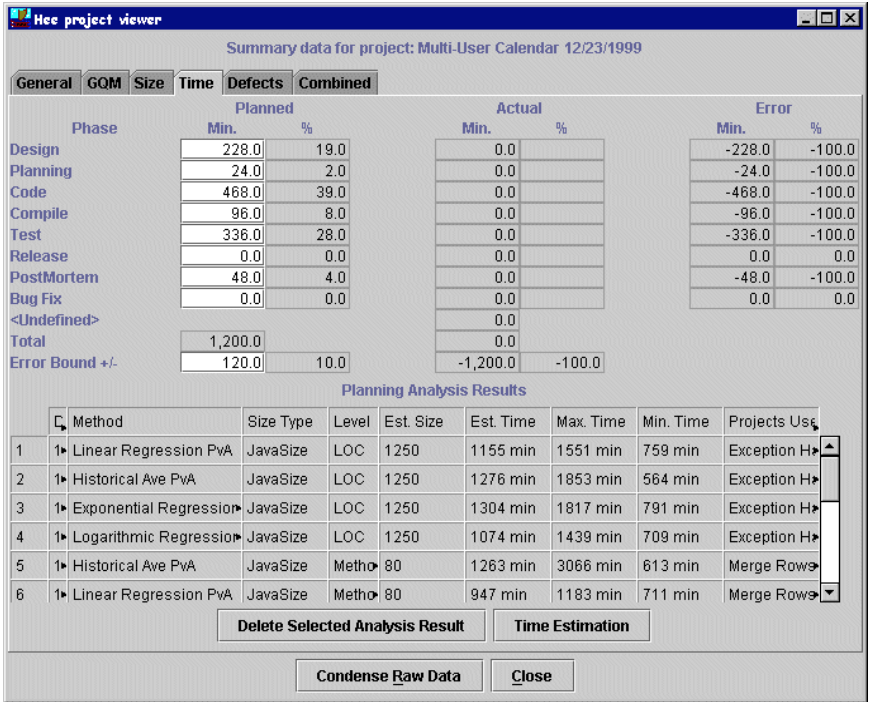
**2.1 Leap:**

It was found that on occasion, the PSP would make wrong conclusions about the development process leading to data quality problems. A toolkit named LEAP (lightweight, empirical, anti-measurement dysfunction and portable software process measurement) toolkit was developed to avoid the data quality problem with the PSP manual. The main focuses of LEAP were;

1. **Lightweight** - The toolkit does not create additional work for the software engineer.
2. **Empirical** - The software engineer could improve his process by learning from his personal experiences
3. **Anti-measurement dysfunction** - LEAP ensures that the developer does not alter his process based on measurement dysfunction and subsequently, altering his process in a way that would hinder it.
4. **Portable** – software developers can bring all their data and tools they use with them from organization to organization.

(Moore, n.d.)

The user as before manually enters their data using PSP methodology and the LEAP then analyses the PSP data for the user therefore improving the data quality, decreasing manual analysis and the work involved for the user. It can also perform analyses that the PSP does not provide such as regression analyses (Johnson, 2013).



(Moore, n.d.)

Figure 2: This is a screenshot from the LEAP toolkit. It is a summary of a project including analyses.

**Limitations of LEAP:**

However, developers of LEAP quickly came to agree with Humphrey’s earlier statement that it was not possible to completely automate the analysis of the PSP data. If one project differs significantly from another project, then new components may be needed for the LEAP tookit. Adding new components would be time costly. The PSP on the other hand simply requires an extra spreadsheet (Johnson, 2013).

**2.2 Hackystat**

The Hackystat was first developed in the University of Hawaii. It was created to acquire and analyze the data from PSP automatically. It collects the 2 main types of software metrics; process and product metrics through sensors on development tools and sends the data to a server. There are 4 design features of the Hackystat:

1. Software developers can store their projects and data privately but also on a server of cloud type.
2. The Hackystat ensured that the data collection was non-obstructive of the developer’s work. The developer does not have to stop what they are doing to record data. The Hackystat does this automatically and unobtrusively.
3. The Hackystat is capable of collecting data by the second allowing it to make accurate insights into the software development process.
4. It is also capable of tracking group-based development. It allows different users to make changes to the same project and can track these changes and developments.

**Limitations of the Hackystat:**

Developers were unhappy with the idea that the Hackystat was collecting their data unbeknownst to them. There were also complaints about how the Hackystat recorded their actions by the second and therefore created a transparency to the developer’s work. Moreover, developers were not always comfortable with the idea that managers had access to all of their data.

(Johnson, 2013)

**2.3 PROM**

PROM was developed by the university of Bolzano researchers to allow software engineers to keep their projects under control. It is similar to the Hackystat in that it automatically collects data and analyses the software metrics of the product and development process to allow the developer to improve his process. PROM acquires data from the software engineer, his/her workgroup and the entire enterprise entity. It ensures privacy for the developer, only revealing his data that they choose to share. There are 4 main core elements included in PROM;

1. The PROM database stores collected data from the development process.
2. The PROM server has an interface to the PROM database. The server is an online service that enables the user to view acquired data and perform data analysis specific to their project.
3. PROM uses plug-ins that records the data and time log of the actions of the developer and sends it to the plug ins server along with identification of the user from whom the data was recorded from. The plug in feature allows multiple developers to work on the same project.
4. The PROM Plug-ins server acquires data from the plug ins, storing them in a cache. The new data is sent to a data analysis WebMetrics tool that calculates software metrics. This new formation is then stored in the PROM database.

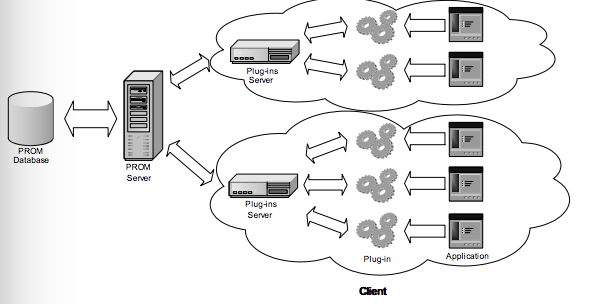


Figure 3: The structure of PROM.

(Sillitti, Janes, Succi and Vernazza, n.d.)

**2.4 Companies who provide data analytics services**

1. CodeClimate

CodeClimate founded in 2011 offers services to developers where they can push their code to the website and CodeClimate will incorporate ‘fully-configurable test coverage and maintainability data’ at all stages of the development process ensuring quality improvement (Codeclimate.com, n.d.).

1. Sixth Sense Analytics

A start-up company Sixth Sense Analytics produces reports for the developer. The service produces systems reports showing metrics such as time invested in the project, team reports showing the focus on each of the activities, individual reports showing details of the actions of the individual and community reports revealing analyses of metrics such as patterns of activity. It was founded in 2004 and began using Hackystat technology for their commercial services in 2006 (Johnson, 2013).

1. Codebeat – takes into account metrics such as cyclomatic complexity, number of returns within a function and the number of arguments of functions and analyses them to ensure the quality software.
2. Codacy
3. Covarity
4. Scutinizer

**Section 3:**

**The algorithmic approaches available to analyze the data from the software engineering process**

**3.1 MSR**

In recent times, mining software metrics has become highly useful in improving the software engineering process. Mining Software Repositories (MSR) field works by analyzing data from software repositories. MSR has 5 types of repositories; Source Control repositories, Bug repositories, Archived Communication, Deployment logs and Code repositories. Software metrics can then be mined from software repositories that can be used to help improve the software engineering process. MSR can also be used to advise on modifications to code and predict the quality of the code. Microsoft have been known to mine metrics from 5 different software systems in order to predict future software failure or software quality problems (Blog.inf.ed.ac.uk, n.d.).

**3.2 Software Intelligence**

The term ‘Software Intelligence’ (SI) describes the information presented to software practitioners about their project at all stages of the development process. The information presented are facts about the current state of their software and enables them in their decision making process. It aids software practitioners in planning and gives them an awareness of the limitations and quality of their software. MSR is enabling SI in the software engineering industry. Below are statements on how SI is currently being used and how it can be enabled widespread in the future

Currently, there are huge amounts of research and MSR work done on source control repositories and bug repositories. Past MSR published papers mainly focus on how to assist the developers. In order for SI to become a reality, MSR work should expand into the other remaining repositories and suggest ways to assist other people who work with software during its lifecycle, example: managers and testers.

“SI is more than just helping with coding”

Currently, people associate the MSR field with utilizing solely historical data. To promote SI in the future, people will have to make use of all types of data, for example, notes from meetings of software practitioners and online postings about software.

“SI should leverage all types of repositories not just historical ones”

Currently, MSR work focused on using previous set data mining (DM) algorithms even if they did not fit their software. In the future, SI can be encouraged by adopting and adapting DM algorithms and develop new ones.

“SI and DM field should work closer”

(Hassan and Xie, n.d.)

**3.3 Machine Learning: Supervised and Unsupervised methods:**

In the development of software systems, it is essential that the software produced is of a good quality. Therefore, algorithms that can predict the quality of software using software metrics are essential. This can be done by using machine learning and data mining techniques (Dick et al., 2004). There are both supervised and unsupervised methods of machine learning for determining the quality of the software.

Supervised methods usually involve predicting the quality using software metrics and historical data. Unsupervised methods are capable of predicting the quality of the software without historical data, using only software metrics. These methods are useful in software projects when there is little knowledge and experience of the software in question recorded.

**Supervised Learning Methods**

Classification is an example of a supervised method of machine learning that can be used in measuring the software engineering process. It is useful in the detection of bugs. Software modules can be classified as defective or non-defective. They are classified by software complexity metrics such as size of the code and McCabe’s cyclomatic complexity.

Other supervised methods include linear regression, decision trees and Naïve Bayes classifier.

(Aleem, Capretz and Ahmed, 2015)

**Unsupervised Learning Methods**

A common unsupervised method of predicting software quality is cluster analysis. Cluster Analysis groups data points according to their software metrics values. The idea is based on the idea that failure-prone software modules will have similar software metrics. Clusters can then be classified by their quality rather than classifying each individual software modules.

(Zhong, Khoshgoftaar and Seliya, n.d.)

Other unsupervised methods of machine learning used in software metrics are the Association rule mining which looks for patterns in the data, K-means and K nearest neighbor.

**Fuzzy Cluster Analysis, an unsupervised learning method:**

Fuzzy Cluster Analysis is an unsupervised learning method. In general cluster analysis looks for similar structures and patterns among data points and groups them according to similarities. Data points within a cluster are similar and clusters are dissimilar from each other. This is a simple cluster analysis in which humans can easily interpret. Fuzzy cluster analysis allows for ambiguity in that it allows data points to partly belong to multiple groups instead of the traditional method of cluster analysis where a data point must belong to one group.

The algorithm that Fuzzy Cluster Analysis is based on is Fuzzy c-Means. The algorithm works iteratively identifying clusters that minimize its cost function. This algorithm is classified as an unsupervised method as it looks for patterns among the data rather than clustering by their actual classification. This algorithm can then classify points into clusters of riskiness and focus in on groups that a high average metric value. This can be done before any failures have occurred making it an unsupervised learning method (Dick et al., 2004).

**3.4 How well can these algorithms address the questions of the software engineer?**

* I find that complications may arise when algorithms mine software databases from the internet. Anyone can write anything online whether it is true or false. Algorithms could then use false data leading to wrong conclusions and could undermine the quality of the software engineering development process and not address the questions of the software engineer effectively.
* Using algorithms that make use of cluster analysis can also compromise results. Software modules could be forced to fit clusters that may not be accurate.

**Section 4:**

**Ethics**

With the development of Software Intelligence and Data Mining techniques, comes the question of ethics. Ethics in the software engineering process addresses the issues of accuracy, data privacy, data sovereignty and the use of AI in decision making processes.

**4.1 Ethics concerning Accuracy:**

1. In the development process, there is a need to predict future states. If the software is complex, ethical issues exist regarding accuracy. Past experience has shown that it makes more sense to work around an error that has occurred rather than attempt to fix it as this will result in an increased number of errors. Ethical dilemma arises when systems are developed that are used in decision making processes that affect humans but they are not entirely accurate because it was deemed better to work around the error than try fix it during the software development process.
2. If a data miner makes a mistake, this could lead to ethical issues. If the data miner for example selects the wrong problem for data mining, ignores unusual or suspicious patterns in the data or does not adequately study the data and its results, they could come to an incorrect conclusion which would wrongly influence a decision making process. This in turn could affect humans.
3. Accuracy can also be hindered when choosing what information, formulas, indicators or models to use in the software development process. For example, in 1991, a study was made into farmer’s needs. ‘Soil fertility’ was used as an indictor leading to deceptive and unsatisfying results. The indictor they should have been looking into was ‘duration of fallow’. This was only discovered after 2 years of research. If the wrong indictor or model is chosen, this could lead to inaccurate results.
4. There is also the question of accuracy when it comes to interpreting the data. Results can be presented in a way to deliberately gave an individual a false perception of the results. Ethical issues arise here if one was to intentionally mislead an individual with the data acquired.

(Thomson and Schmoldt, n.d.)

**4.2 Ethics concerning data sovereignty:**

I find that the main ethical issue concerning data sovereignty in today’s world is that people are unaware of the consequences of providing their information and unaware of where and what it could be used for. The question also arises that if someone makes information freely available, for example on their public social network profile, who owns this data and is it ethical to use it for data mining purposes.

**4.3 Ethics concerning AI:**

AI can be divided into 2 waves. The first wave of AI involved getting machines to think like humans. The 2nd wave can be considered as developing machines that can outperform humans but in a non-thinking way. People have the misconception that the machines are incapable of empathy, creativity, non-routine work or making moral judgements. There are 2 reasons why this is a misconception:

1. If you decompose ‘non-routine’ work, routine work can be identified in the broken down processes.
2. The **AI fallacy** is believing that machines need to be developed to replicate the thinking processes of human experts. However, machines can be developed that can handle ambiguity not by teaching them how to judge a situation like a human but by doing so using algorithms. For example, an expert chess player uses his judgement to make his next move. AI would use a brute force algorithm and can identify 300 million different solutions in under a second.

Ethics becomes as issue when AI is used to make the decision should as;

1. Passing a life sentence
2. Turning off a life support machine

(Susskind and Susskind, 2015)

In my opinion, these are questions that should never be decided by a machine. As explained in 4.1, there are accuracy issues when developing software. If these accuracy issues exist in the software used to answer these questions, this becomes an ethical issue.

**4.4 Ethics concerning data privacy:**

The issues with privacy can create problems for software design, development and deployment.

1. The topical issue concerning ethics with data mining when analyzing the software engineering process is the concept that the individual is not aware that information is being collected about them and that information is being using without their knowledge and without their consent.
2. Ethical dilemmas arise when the individual’s data is used improperly or the information they supplied is used for an alternative purpose (van Wel and Royakkers, 2004). With data mining algorithms such as cluster analysis comes group profiling and there is a risk that the analysis is not accurate and would lead to improper use of the individual’s data.

Below are examples of how data mining is being used in the public and private sector where it is unclear whether the practices are ethical;

**In the pubic sector:**

Organizations use public information for a number of different uses. Public sector databases in the US including ‘Marriage Fraud Amendment System’ and ‘Student and Schools System’ are used widely in a number of different government organizations. The Transportation Security Administration (TSA) uses data mining and their factual data such as name, address and birthday to identify terrorists who would subsequently go through additional security. Using data mining techniques, it is expected the number of false positives will decrease significantly. However, with this type of data mining comes data privacy issues and ethical issues. This type of data mining will almost certainly invade an innocent person’s privacy. Moreover, their information is being used for a secondary purpose without the individual’s knowledge (Cook, 2005).

**In the private sector:**

In the private sector, data mining is used to target advertising and cost structures at certain customers. The method by which they use could be described as discrimination. If they are not discriminating by age, race or gender, this practice is considered legal. This raises the issue again, is this practice ethical? Is it ethical to use all of the information of the consumer while they are unaware of its use? I agree with the case that this type of data mining is in the consumer’s interest. If a business cannot participate in target advertising, they would have to invest majorly in huge advertising campaigns which would drive up costs for the consumer. Therefore, it can be seen that data mining can have benefits and advantages for the consumer (Cook, 2005).

**4.5 Closing Opinion**

A possible solution is making people aware about who can have access to data that they are providing. Once the person is made aware who has access to that data, they will in turn have an idea of what their data may be used for. In most cases, as mentioned earlier, it will benefit them;

* they will be targeted with advertisements and discounts that apply to them
* employers can use the data to improve employee’s lives at work
* software developers will be able to develop software of a better quality if they can data mine an increased amount of data during the software development process.

With regards to data being collected unbeknownst to the individual, they should realize that they are one in millions and their data is not being studied on an individual level but being studied as part of a cluster.

I also find that if people make more of their data available, data mining algorithms of software repositories will become increasingly accurate and the worries of data fidelity will be reduced.

However, I firmly believe that decisions that have significant impacts on the lives of people such as turning of life support or passing a life sentence, as mentioned earlier should never be decided solely by a machine. They can advise but fundamentally, a human should make the final call on such questions.

References:

Aleem, S., Capretz, L. and Ahmed, F. (2015). Benchmarking Machine Learning Techniques for Software Defect Detection. International Journal of Software Engineering & Applications, 6(3), pp.11-23.

Blog.inf.ed.ac.uk. (n.d.). Improving the software development process by Mining Software Repositories – SAPM: Course Blog. [online] Available at: https://blog.inf.ed.ac.uk/sapm/2014/02/14/improving-the-software-development-process-by-mining-software-repositories/ [Accessed 24 Nov. 2017].

Codeclimate.com. (n.d.). About Code Climate. [online] Available at: https://codeclimate.com/about [Accessed 23 Nov. 2017]

Cook, J. (2005). Ethics of Data Mining. [online] http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1443&context=article. Available at: http://Ethics of data mining [Accessed 24 Nov. 2017].

Dick, S., Meeks, A., Last, M., Bunke, H. and Kandel, A. (2004). Data mining in software metrics databases. Fuzzy Sets and Systems, 145(1), pp.81-110.

Fenton, N. and Neil, M. (1999). Software metrics: successes, failures and new directions. Journal of Systems and Software, 47(2-3), pp.1-2.

Hassan, A. and Xie, T. (n.d.). Software Intelligence: The Future of Mining Software Engineering Data. [online] Nitrd.gov. Available at: https://www.nitrd.gov/Subcommittee/sdp/foser/Software%20Intelligence%20The%20Future%20of%20Mining%20Software%20Engineering%20Data%20p161.pdf [Accessed 24 Nov. 2017].

Johnson, P. (2013). Searching under the Streetlight for Useful Software Analytics. IEEE Software, 30(4), pp.57-63.

Moore, C. (2017). https://pdfs.semanticscholar.org/3c3f/6302a52b6b42ac8587545b0c6ddeb70f6e88.pdf. [online] Available at: https://pdfs.semanticscholar.org/3c3f/6302a52b6b42ac8587545b0c6ddeb70f6e88.pdf [Accessed 20 Nov. 2017].

Resources.sei.cmu.edu. (n.d.). Cite a Website - Cite This For Me. [online] Available at: https://resources.sei.cmu.edu/asset\_files/SpecialReport/2009\_003\_001\_15029.pdf [Accessed 19 Nov. 2017].

Sillitti, A. Janes, G. Succi, and T. Vernazza, “Collecting, integrating and analyzing software metrics and personal software process data,” in Proceedings of the 29th Euromicro Conference. IEEE, 2003, pp. 336– 342.

Sommerville, I. (2008). Software productivity. [online] Ifs.host.cs.st-andrews.ac.uk. Available at: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Planning/productivity.html [Accessed 19 Nov. 2017].

Stepford, B. (2017). Do Metrics Have a Place in Software Engineering Today? - ben stopford. [online] ben stopford. Available at: http://www.benstopford.com/2004/03/14/do-metrics-have-a-place-in-software-engineering-today/ [Accessed 19 Nov. 2017].

Susskind, R. and Susskind, D. (2015). The future of the professions: how technology will transform the work of human experts.

Thomson, A. and Schmoldt, D. (n.d.). Ethics in computer software design and development. [online] Srs.fs.usda.gov. Available at: https://www.srs.fs.usda.gov/pubs/VT\_Publications/01t7.pdf [Accessed 24 Nov. 2017].

van Wel, L. and Royakkers, L. (2004). Ethical issues in web data mining. Ethics and Information Technology, 6(2), pp.129-140.

Yano, K., Ara, T., Watanabe, J., Sato, S. and Moriwak, M. (2015). Measuring Happiness Using Wearable Technology. [online] Hitachi.com. Available at: http://www.hitachi.com/rev/pdf/2015/r2015\_08\_116.pdf [Accessed 23 Nov. 2017].

Zhong, S., Khoshgoftaar, T. and Seliya, N. (n.d.). Unsupervised Learning for Expert-Based Software Quality Estimation. [online] Ai2-s2-pdfs.s3.amazonaws.com. Available at: http://ai2-s2-pdfs.s3.amazonaws.com/bcf3/9cc6aeaba6489e10042bbb38cdd49110f984.pdf [Accessed 24 Nov. 2017].